A Model for Executing Multidisciplinary and Multizonal Programs

Eric Barszcz[*] Sisira K. Weeratunga[†] Eddy Pramono[†]

NAS Applied Research Branch
NASA Ames Research Center
Moffett Field, CA 94035

Report Number: RNR-93-009

March 26, 1993

Abstract

This paper presents the model of computing multidisciplinary and/or multizonal applications on parallel machines that is being pursued at NASA Ames Research Center. The model describes execution on a parallel machine with some number of processors, each with local memory, connected by a network. It does not discuss processor rates, size of memory, network latency, network bandwidth, network connectivity, disk space, I/O bandwidth or mass storage. It also does not discuss language and discipline specific algorithm issues. Model rationality and functionality are presented.

[*]The author is an employee of NASA.

[†]The author is an employee of Computer Sciences Co. This work was funded through NASA contract NAS 2-12961

1 Introduction

The problem domain of particular interest to NASA Ames Research Center is aeronautical research and engineering. Some of the disciplines involved in modeling an aircraft though its flight envelope are fluid dynamics, structural dynamics, thermal analysis, propulsion, control theory, electromagnetic analysis, and acoustics. All of the computational aeroscience (CAS) grand challenges of the HPCC program involve coupling two or more of these disciplines.

In addition, geometrically complex aircraft configurations, often with one or more components in relative motion with respect to others, require composite mesh methodologies for efficient spatial discretization [4]. A multizonal application meshes individual components of the configuration separately and forms a composite mesh by overlapping component meshes. The overlapping meshes interact through data interpolation at inter-mesh boundary points [7].

Currently, there are no applications that combine all of the disciplines. However, several codes exist that combine at least two of these disciplines. Some examples in the aeronautical domain are fluid-thermal, fluid-structures, fluid-acoustics, and fluid-electromagnetic. At NASA Ames Research Center, fluid-thermal [12], fluid-structures [9], and multizonal fluid dynamics [10] codes have been implemented on a message passing parallel computer, the Intel iPSC/860.

This paper presents the model of computing multidisciplinary and/or multizonal applications on parallel machines that is being pursued at NASA Ames Research Center. The model describes execution on a parallel machine with some number of processors, each with local memory, connected by a network. It does not discuss processor rates, size of memory, network latency, network bandwidth, network connectivity, disk space, I/O bandwidth, or mass storage. It also does not discuss language and discipline specific algorithm issues.

The paper has three sections. Section 2 describes general multidisciplinary problem characteristics. Section 3 discusses possible solution approaches. Section 4 presents three computational models and their requirements.

2 Problem Characteristics

A multidisciplinary problem is composed of a collection of two or more interacting disciplines. A given discipline may interact directly with one or more of the other disciplines. Also, these interactions occur simultaneously across all interdisciplinary boundaries. Similar remarks apply to component meshes in a multizonal computation.

A multidisciplinary problem is naturally decomposed into different interacting disciplines, each of which may model widely different physical phenomena. Therefore, the equations used to describe the underlying physics (PDEs/ODEs and/or integral equations) vary between disciplines. As a result, the computational techniques used for computer simulations may have widely different numerical algorithm characteristics, data structures, and computational requirements. In addition, on parallel computers with physically distributed memory, the above differences may induce distinct data partitioning strategies for each discipline. For example, some disciplines may be modeled using finite difference discretization on logically structured meshes while others may use a finite element discretization on unstructured meshes. Also, the computational requirements within each discipline may vary dynamically due to solution adaptive mesh refinement/coarsening, moving boundaries, and evolving material non-linearities, etc.

3 Solution Approaches

Given a multidisciplinary and/or multizonal problem, what would be the best approach to solving it on a parallel computer? On a serial computer, the disciplines and/or component meshes must be processed serially. This is accomplished by having a doubly nested loop. The outer loop iterates for the appropriate number of time steps and the inner loop iterates over the disciplines (meshes).

For a parallel computer, the disciplines (meshes) may be executed in parallel or sequentially. In the parallel execution mode, each discipline (mesh) may be partitioned across a different set of processors and computation proceeds in parallel within disciplines (meshes) and across disciplines (meshes). Both data parallelism and task parallelism are used. In the sequential execution mode, each discipline (mesh) is partitioned across all processors and the computation proceeds in parallel within disciplines (meshes) but sequentially across disciplines (meshes). Only data parallelism is used.

The decision whether or not to use task parallelism across disciplines (meshes) is influenced by several factors: memory requirements, memory efficiency, computational requirements, Amdahl's Law, software engineering issues, and multidisciplinary coupling issues.

In the discussion below when comparing two approaches, a fixed number of processors and a fixed problem size are assumed. A fixed number of processors is a valid assumption because at any instant, the goal is to use the available resources most efficiently. A fixed problem size is appropriate because once the relevant physics is resolved to the desired level of accuracy, further refinement "wastes" computational resources (memory and CPU cycles).

3.1 Memory Requirements

Memory requirements include the size of the executable and the size of the data. For a SIMD machine, there is a single copy of the executable and the disciplines are processed sequentially (data parallelism only). On a distributed memory MIMD machine, there are generally multiple copies of the executable. If the disciplines are processed in parallel (task and data parallelism), the set of processors associated with a discipline only need the executable for that discipline. If the disciplines are processed sequentially (data parallelism only), each processor needs a copy of the executable for all disciplines. Depending on the amount of local memory, whether or not virtual memory is supported, and the parallel I/O bandwidth, this may or may not be an issue. Processor utilization degrades with the number of processes contending for I/O resources. Therefore, most massively parallel processors should be thought of as physical memory machines and are considered such in this paper.

The memory required to store data also varies. On a SIMD machine, since the amount of data for each discipline (mesh) vary, some data structures may be padded depending on specific machine or compiler requirements [11]. Also, temporary variables are often the size of the whole mesh. On a MIMD machine, when a domain is partitioned across multiple processors, some data are often duplicated to save communication costs [2]. Also, temporary variables, typically scalar or vector temporaries, and small global data structures (i.e., small look up tables) are replicated to avoid bottlenecks. Buffer space is also needed on each processor to buffer messages in a message passing environment.

If the disciplines (meshes) are processed in parallel, each discipline (mesh) has a better surface to volume ratio for its data implying less duplicated data. (Assuming a fixed number of processors for the whole application.) Also, the memory required for temporary variables depends upon the individual discipline. If the disciplines (meshes) are processed sequentially, the local memory associated with a processor must be shared by all disciplines (meshes). This implies a higher surface to volume ratio for each discipline (mesh) since each discipline (mesh) is spread over more processors. This in turn implies more duplicated data to avoid communication. Also, the memory requirement for temporary variables is dependent on the discipline (mesh) with the largest requirements.

3.2 Memory Efficiency

Greater parallelism implies more memory is required to solve the application efficiently. Let memory efficiency be defined as

$$\varepsilon_M = \frac{M_0}{M_N},$$

where ε_M is the memory efficiency, M_0 is amount of the memory required to fit the problem on the smallest number of processors (P_0), and M_N is the amount of memory used on P_N processors ($P_0 \leq P_N$). Then the claim is that memory efficiency decreases as the number of processors increase. All that is necessary for this claim to be true is for a single variable/compiler temporary to be replicated across two or more processors.

As an example of how memory requirements can grow with increased parallelism, consider a $32 \times 32 \times 32$ mesh. (Fairly small mesh sizes occur frequently in multizonal applications.) Form an independent scalar tridiagonal system out of each column. Then there are 32^2 independent tridiagonal systems, each of length 32. Compare the memory requirements to solve the systems on a serial machine, a single vector processor, and a parallel vector machine. The solution technique will be Gaussian elimination without pivoting where the input may be overwritten.

On a serial machine, $32^3 + 3 \times 32$ words are required, 32^3 for the right hand side and 3×32 for a single tridiagonal system. The systems are formed and solved one at a time.

On a single vector processor, $32^3 + 3 \times 32^2$ words are required, 32^3 for the right hand side and 3×32^2 to form 32 independent tridiagonal systems. At least 32 of the systems are formed at a time to allow for vectorization across independent systems.

On a 32 node parallel vector machine, 4×32^3 words are required, 32^3 for the right hand side and 3×32^3 to form 32^2 independent tridiagonal systems. Thirty two of the systems are formed at a time on each processor to allow for vectorization.

If the serial machine has a memory efficiency of 1.0, then the single vector processor and parallel vector machine have memory efficiencies of 0.92 and 0.25 respectively.

3.3 Computational Requirements

Computational requirements for different disciplines and different meshes vary. Also, mesh sizes may vary widely in multizonal computations yielding very different computational requirements for different meshes. This leads to load balancing issues. For example, computational requirements in the controls discipline is much smaller than the computational requirements to compute a fluid flow field.

If disciplines (meshes) are processed in parallel (task and data parallelism) then the number of processors assigned to each discipline (mesh) can be adjusted to computationally load balance the application. (Subject to memory constraints.)

If disciplines (meshes) are processed sequentially (data parallelism only), for disciplines (meshes) with small memory requirements and small computational requirements, trying to use all available processors may actually slow down the application due to the decrease in the computation to communication ratio. If the computation associated with a discipline (mesh) is unable to use all available processors, then some processors will be idle during that phase of the computation.

Hockney and Jesshope [6] discuss trade offs in algorithm selection. Which algorithm is "best" depends upon the machine architecture, actual number of processors being used, and the problem size. An algorithm with more inherent parallelism might not execute as fast as an algorithm with less inherent parallelism due to an overall increase in operations.

3.4 Amdahl's Law

Efficiency also varies depending on whether the disciplines (meshes) are processed in parallel or sequentially. Amdahl's Law says that speedup is bounded by the fraction of the code that is serial:

$$S \leq \frac{N}{(N-1)\nu + 1},$$

where S is the speedup, N is the number of processors, and ν is the fraction of the code that is serial. Efficiency, ε, is defined as

$$\varepsilon = \frac{S}{N} = \frac{1}{(N-1)\nu + 1}.$$

Amdahl's Law is relevant since these discussions compare and contrast the processing of a fixed number of multiple disciplines (meshes).

If the disciplines (meshes) are processed in parallel, assume that the number of processors assigned to each discipline (mesh), G_i, is such that there is near perfect load balance across disciplines (meshes). Given the same number of processors ($N = \sum G_i$), processing disciplines (meshes) sequentially must have a lower efficiency. Each discipline (mesh) is spread over more processors and Amdahl's Law implies efficiency decreases as the number of processors increases for a fixed problem size.

For example, assume that a multizonal application has two identical sized meshes with the same computational load for each. Further, assume that the computation on each is 99.9% parallel. Given 1000 processors, if the meshes are processed in parallel (500 processors assigned to each mesh) then the efficiency is 66.7%. If the meshes are processed sequentially (1000 processors assigned to each mesh) then the efficiency is 50.0%. This means that processing the meshes sequentially will take 33% longer than processing them in parallel.

Other sources of inefficiency are an increased communication to computation ratio and an increase in redundant computations as the percentage of duplicated data increases.

3.5 Software Engineering

Some of the software engineering issues are modularity, independence, extensibility, modifiability, readability, and maintainability. Software engineering may be thought of as complexity management. Two primary ways humans deal with complexity are divide-and-conquer and abstraction. The trend over the last thirty years in software engineering has been towards greater code modularity and greater information hiding. Structured programming, top-down design, program modules, abstract data structures, object oriented programming are all examples of this trend. They encourage the partitioning of problems into simpler pieces, interfaces to be defined, and the implementation details to be hidden behind the interfaces.

3.6 Multidisciplinary Coupling

Three principal methods for coupling multiple disciplines are global explicit integration, global implicit integration, and partitioned analysis [5]. Global explicit integration applies an explicit time integration algorithm to the global set of equations. However, stability is a concern and the time step must be chosen based on the discipline with the stiffest requirements.

If a global implicit coupling scheme is used, the problem can be expressed in block matrix form where the main diagonal blocks represent the individual disciplines and the off-diagonal blocks represent the coupling between the disciplines. Then the whole system can be solved as a large single matrix. However, in many instances, it may not be possible to directly couple two disciplines. This is due to difficulties associated with the explicit evaluation of off-diagonal blocks. For example, it would be difficult to directly couple a particle simulation of a rarefied gas to the structural and thermal analysis of the body over which it flows. Even if all off-diagonal coupling terms can be evaluated explicitly, direct inversion of such a large sparse matrix would be prohibitively expensive in terms of both memory and CPU time for problems of any consequence to the aeronautics community. As a result, it is necessary to resort to some form of iterative solution approach based on the concept of sub-structuring or partitioned analysis. A natural form of sub-structuring across discipline boundaries would be inevitable due to widely different numerical characteristics across disciplines. A monolithic iterative solver is unlikely to be able to cope with all the diverse disciplines in an efficient manner.

In partitioned analysis, the effect of off-diagonal blocks are represented as forcing functions on the right hand side. The computation of these forcing terms would induce inter-disciplinary communication. Partitioned analysis leads to block iterative methods (block Jacobi, block Gauss-Seidel, etc.). Also, with partitioned analysis, it is easier to extend the application to include new disciplines. However, robust and efficient multidisciplinary and/or multizonal coupling techniques is an active area of research.

Park and Felippa [8] list many of the problems associated with global implicit coupling and the advantages of partitioned analysis. From a software engineering viewpoint, the list of disadvantages of direct coupling and advantages of partitioned analysis could be said about monolithic programming versus modular programming respectively.

4 Computational Models

Given the above considerations (memory requirements, computational requirements, Amdahl's Law, software engineering, and multidisciplinary and/or multizonal coupling) the most "natural" approach to implementing a multidisciplinary and/or multizonal application on parallel processor is partitioned analysis using a MIMD style of execution across disciplines (meshes). This allows numerical algorithms and programs pertaining to individual disciplines to be developed and tested independently. After each program has been validated, an additional module is added to each that deals with the required interdisciplinary (interzonal) communication.

Several models of computation support the MIMD style of execution across disciplines. Which one is best depends upon the physical problem, solution algorithm, hardware resources, and operating system support available. Below we present three models and describe the functionality that is desired from each.

The discussions assume the following definitions: NP equals the total number of processors used, a group G_i is the set of processors assigned to a particular discipline, NG is the total number of groups, and GS_i is the number of processors in group G_i.

In all of the models, it is assumed that the user has the option to control the set of physical processors assigned to a group and the data layout on that set of processors.

4.1 Static Model

In the static model, all resource requirements are known at startup (i.e., NP, NG, and GS_i are all known). Functional requirements are that the number of processors assigned to each discipline is independent of the other disciplines, a different executable may be loaded into each group, each group has its own logical numbering, global operations such as SUM or MAX apply to the local group, and some mechanism is provided to establish communication between an individual processor in one group and an individual processor in another group.

The logical numbering within a group, global operations applying only to the local group, and different executables all support independent development and testing of disciplines. The independent assignment of processors to disciplines allows the application to be statically load balanced. Point to point communication between groups allow coupling information to be passed directly between the processors that need to know the information.

Groups are neither created nor deleted dynamically. Everything is initialized at startup and continues until application termination. Currently, there are several applications [10, 9, 3] using this model at NASA Ames Research Center based on the intercube communication software developed for the Intel iPSC/860 [1].

There are several problems with the iPSC/860 implementation of the static model. Debuggers do not work for multiple independent groups of processors. Groups are restricted to a power-of-two number of processors and only one group may be assigned to a processor thereby limiting the ability to load balance the computation. Also, existing batch job submission software, Network Queuing System (NQS), does not support this style of computation.

4.2 Dynamic Model

In the dynamic model, groups may be created or deleted dynamically. NP and NG vary dynamically. GS_i is fixed at group creation. The functional requirements are (1) the number of processors assigned to each discipline is independent of the other disciplines, (2) a different executable may be loaded into each group, (3) each group has its own logical numbering, (4) global operations apply to the local group, (5) mechanisms for group creation and deletion are needed, and (6) some mechanism to establish communication between an individual processor in one group and an individual processor in another group.

The reasoning for the functionality is the same as in the static model with the additional requirements of dynamic creation and deletion of groups. The size of an individual group is fixed at group creation. If a group needs to increase its size, a new group may be created and the relevant information copied from the old group. Then the old group name is reassigned to the new group and the old group deleted. The create, copy and delete approach to dynamic group size is acceptable because disciplines often assume a particular data to processor layout. Adding processors to the current group would also require a data reorganization. Moving to a new set of processors may ease the process. This form of process migration requires that the change in location of the group be transmitted to all groups that need to know.

4.3 Hybrid Model

In the hybrid model, the total number of processors used by the application, NP, is fixed at application startup. However, the number of groups, NG, and their sizes, GS_i, vary at runtime. Functional requirements are (1) the number of processors assigned to each discipline is independent of other disciplines, (2) a different executable may be loaded into each group, (3) each group has its own logical numbering, (4) global operations apply to the local group, (5) mechanisms for group creation and deletion are needed, and (6) some mechanism to establish communication between an individual processor in one group and an individual processor in another group.

Having a fixed number of processors assigned to the application simplifies some problems and creates others. Keeping the group information consistent across groups is simplified because it is a bounded problem and tables may be kept on a processor basis rather than a group basis. However, what happens when a group is created and there are no idle processors? Possible solutions are to either abort the application, inform the application that it is out of processors, or allow multiple groups per processor and time slice and share local memory (memory slice) across groups. Recall that parallel processors are being considered physical memory machines.

The most flexible solution is informing the application that it has run out of idle processors and let it decide if it wants to memory slice or abort. If memory slicing is selected, the user should have the option to specify a logical processor layout and the set of physical processors that are to be memory sliced. This implies the ability of an application to interrogate the state of any processor.

With memory slicing, it is also desirable to have group migration as an option. This is an issue where solution adaptive algorithms are used. Assume that a region needs refinement. Then a new grid with greater refinement can be placed over the region. The new grid is treated as a logically separate group. In other places, the grid may be coarsened. In those places, groups terminate. As groups terminate, processors may become idle while other processors are memory slicing. Under these conditions, the user may want groups to migrate to the idle processors.

One of the advantages of the hybrid model is that it could be submitted as a batch job. The resources required are fixed and specifiable at application instantiation.

5 Conclusion

Based on memory requirements, memory efficiency, Amdahl's Law, computational requirements, computational efficiency, software engineering, and multidisciplinary (multizonal) coupling a computing model that supports a MIMD style of computing across disciplines is the most natural.

Three computational models were introduced. The static model is currently being used at NASA Ames for multidisciplinary and multizonal applications. We expect to move towards the hybrid model in the next year to support solution adaptive methods.

References

[1] E. Barszcz. Intercube Communication for the iPSC/860. In *Scalable High Performance Computing Conference '92 proceedings*, pages 307–313, April 1992.

[2] E. Barszcz, T. F. Chan, D. C. Jespersen, and R. S. Tuminaro. FLO52 on Hypercubes: Performance and Modelling of a Multigrid Euler Code. *International Journal of High Speed Computing*, 1(3):481–503, 1989.

[3] E. Barszcz, S. K. Weeratunga, and R. L. Meakin. Dynamic Overset Grid Communication on Distributed Memory Parallel Processors. To be presented at 11th AIAA Computational Fluid Dynamics Conference, Orlando Florida, July 1993.

[4] J. A. Benek, P. G. Buning, and J. L. Steger. A 3-D Chimera Grid Embedding Technique. American Institute of Aeronautics and Astronautics paper 85-1523, July 1985.

[5] C. A. Felippa and T. L. Geers. Partitioned Analysis for Coupled Mechanical Systems. College of Engineering and Applied Science, University of Colorado, Boulder, CO 80309.

[6] R. W. Hockney and C. R. Jesshope. *Parallel Computers.* Adam Hilger, Bristol and Philadelphia, 2nd edition, 1988.

[7] R. L. Meakin. A New Method for Establishing Inter-Grid Communication Among Systems of Overset Grids. American Institute of Aeronautics and Astronautics paper 91-1586, June 1991.

[8] K. C. Park and C. A. Felippa. Partitioned Analysis of Coupled Systems. In T. Belytschko and T. J. R. Hughes, editors, *Computational Methods for Transient Analysis*, chapter 3, pages 157–219. Elsevier Science Publishers B.V., North Holland, Amsterdam, New York, and Oxford, 1983.

[9] E. Pramono and S. K. Weeratunga. Aeroelastic Computations for Wings through Direct Coupling on Distributed-Memory MIMD Parallel Computers. Work in progress at the Applied Research Branch, NAS System Division, NASA Ames Research Center, Moffett Field, CA 94035.

[10] J.S. Ryan and S. K. Weeratunga. Parallel Computation of 3-D Navier-Stokes Flowfield for Supersonic Vehicles. American Institute of Aeronautics and Astronautics paper 93-0064, January 1993.

[11] Thinking Machines Corporation, Cambridge, MA. *CM Fortran Programming Guide*, version 1.0 edition, January 1991.

[12] S. K. Weeratunga. Fluid-Thermal Interaction Modelled on the iPSC/860. Research in progress. NAS Applied Research Branch, NASA Ames Research Center, Moffett Field, CA 94035.